Graph-based Clustering for Detecting Frequent Patterns in Event Log Data

Erika Sy[1], Sam Ade Jacobs[2], Aldo Dagnino[3] and Yu Ding[4]

Abstract— **Finding frequent patterns is an important problem in data mining. We have devised a method for detecting frequent patterns in event log data. By representing events in a graph structure, we can generate clusters of frequently co-occurring events. This method is compared with basic association mining techniques and found to give a "macro-level" overview of patterns, which is more interpretable. In addition, the resulting graph-based clustering output for frequently co-occurring event sets is substantially less than association mining, while providing similar information levels. Therefore, the results are more manageable for practical applications.**

I. INTRODUCTION

Frequent pattern mining was popularized in the mid-1990s by Agrawal et al. [1]. They described an association mining algorithm to find reoccurring purchasing patterns of customers. This problem is often called market basket analysis and attempts to find items that frequently get purchased together, e.g., cereal and milk. Frequent pattern mining is a technique in data mining involved in finding frequently occurring itemsets, subsequences, or substructures [2]. A user-defined threshold determines the minimal frequency to be considered as frequent enough. Frequent pattern mining has many applications such as gene sequencing [3], social network trends [4], and mining data streams [5].

We focus specifically on the application of frequent pattern mining to event log data. Event log data capture monitored changes to the state of a system and record the information concerning "who, what, when, where, why" of a system in the event log database. Therefore, event logs are often manifested in a format containing a list of timestamped events with attributes such as: *EventID*: unique event identifier, *EventName*: descriptive name of event, *EventMessage*: detailed description of event, *EventStartTime*: timestamp of recorded event, etc. Examples of event log data are computer event logs, network history logs, control system alarm logs, and machine maintenance logs. These records are often stored sequentially in a database; as such, event logs can be considered as temporally-ordered sequences.

We want to find frequently co-occurring events in event logs. Events that frequently occur together exhibit regularities within the data. When supplemented with domain knowledge, these patterns can provide useful insight. Consider in a machine where a set of event patterns generated from two sources (e.g., a fan and the battery within a laptop) may indicate dependence within the parts of the machine (e.g., fan malfunction leads to overheating battery). A pattern with a critical event may indicate the root cause of the critical event (e.g. battery failure is a critical event caused by overheating caused by fan malfunction). Hence, finding frequently co-occurring events can be very helpful to the practice of root cause diagnosis or anticipating a failure event and thus staging a preventive action ahead of time.

The unique nature of event logs is best suited to two analysis techniques [2]: association mining and sequence mining; we will review them in more details in Section II. Both approaches can be applied to databases of itemsets, i.e., records containing multiple events. The main difference between the two approaches is that sequence mining considers the temporal-order of items (i.e., events), while association mining does not. One limitation common to both approaches is that they often lead to the finding of too many frequent sets. For instance, when using one dataset that has over 6,000 original events (Dataset B in Section V), an existing method yields 3,000 to 4,500 frequent sets, even though the method covers only about 10% of the original events. With this many frequent sets, it becomes impractical for decision makers to make sense out of them easily.

In this paper, we propose a clustering-based approach to provide results that can greatly reduce the final number of frequent sets and are thus easily interpretable to real-world data. A graph is constructed with two characteristics: events with shorter temporal distance are represented by a shorter graph distance, and events with frequent co-occurrence have a higher weight between their paths. Clustering constrains each event to appear only once among all clusters. Each cluster represents a frequently co-occurring set. In a way, the resulting output synthesizes the results of traditional association mining or sequence mining, while being robust to patterns that appear sparingly within the data.

The rest of the paper unfolds as follows: Section II reviews the related works in association mining and sequence mining. Section III describes the proposed solution approach. Section IV analyzes three different datasets. Discussion of the results follows in Section V. Finally, in Section VI, we conclude the paper.

II. RELATED WORKS

Agrawal et. al [1] first proposed an algorithm for mining association rules in market basket data. Association mining

[1]Erika Sy with the Department of Industrial & Systems Engineering, Texas A&M University, College Station, TX erika.sy@tamu.edu

[2]Sam Ade Jacobs with ABB Corporate Research, Raleigh, NC sam.jacobs@us.abb.com

[3]Aldo Dagnino with ABB Corporate Research, Raleigh, NC aldo.dagnino@us.abb.com

[4]Yu Ding with the Department of Industrial & Systems Engineering, Texas A&M University, College Station, TX yuding@tamu.edu

can be stated as two subproblems: 1) frequent itemset generation and 2) rule generation. Frequent itemsets generation mines a database for all sets of items that occur together within baskets at least a specified number of times. This minimum threshold is a user-defined parameter. Using the frequent itemsets, rule generation produces if-then rules: an antecedent itemset appearing in a basket implies a consequent itemset will also appear in the same basket. Soon afterwards, Agrawal et al. [6] proposed the `Apriori` algorithm, one of the most well-known association mining algorithms. Based on the `Apriori` property: no superset of an infrequent itemset can be frequent. The ideas from [6] spawned many variant algorithms over the years [7]–[11].

Association mining does not explicitly use the temporal nature of data to provide results. But since association mining requires baskets of items as inputs, time could be used while generating the baskets of events. If this preprocessing step is undertaken, the temporal information could have been implicitly considered in the process of association mining.

Closely related to association mining, sequence mining, as suggested by its name, is the frequent pattern mining technique that considers the temporal nature of a dataset. The basic idea of sequence mining is to scan through the temporal ordered data records to find subsequences that constitute frequent itemsets (i.e., items appear frequently in order). The problem of sequence mining is that it is more complicated than association mining precisely because the items in a basket are ordered and time is explicitly modeled. Many association mining algorithms are based on enumeration trees, which can be generalized to sequences [12]. Owing to the enumeration nature and the need to go through a large number of combinations, sequence mining often runs slowly when it is applied to a large dataset, which is usually the case for many of the event log datsets.

To alleviate the computational demand for handling large sized event log datasets, events can be partitioned into baskets using a time window. While such partitions reduce the number of combinations to scan through, the data remains in a sequential fashion for sequence mining.

Nevertheless, the key drawback of association mining and sequence mining approaches when applied to event log data, as it has been attempted before [13]–[15], is that these approaches lead to copious amounts of frequent item patterns, too many for decision makers to make practical use of. We present a different approach here to provide a solution to address this shortcoming.

III. GRAPH-BASED CLUSTERING FOR FREQUENT PATTERN DETECTION

For detecting frequent patterns in an event log, at leats two actions are needed: first the events and their connections should be represented in a model, and then a method is needed to extract events similar enough into a pattern set. This is fulfilled in three steps in our proposed method.

For representing the event connections, many association mining and sequence mining algorithms use enumeration trees [12], requiring a pre-processing basketization step for it to be applicable to event log data. Here we propose to model events in a directed graph instead by exploiting the linear, timeline nature of event logs. The graph is constructed such that events with shorter temporal distance are represented with a shorter graph distance, and events with frequent co-occurrence have a higher weight between their paths.

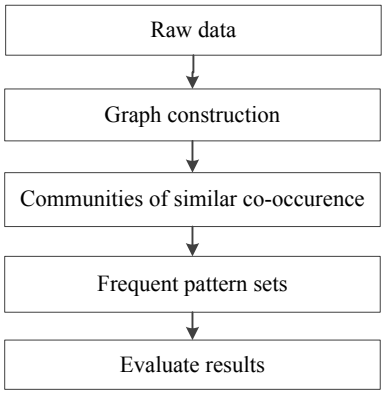

Fig. 1. Proposed methodology pipeline.

Grouping objects based on a similarity measure is the action of clustering. When applied to graphs, clustering is often called community detection [16]. One specific class of community detection algorithms, hierarchical clustering-based, is well-suited to our problem. It does not require prior knowledge about the number of clusters, and it uses connectivity-based similarity metrics. Since the structure of our graph uses path weights and graph distance to represent frequency, the community detection outcomes are the communities of similar co-occurrences.

The last step, a subclustering action, is to filter out the subclusters from the detected communities that have higher enough co-occurrence frequencies. The outcomes of the subclustering is the final frequent pattern sets. A flowchart depicting the steps of the proposed solution is in Fig. 1.

A. Graph Construction

For a timestamped event log file, construct a weighted, directed graph $G(V, E)$ such that:

- Vertices in V represent the unique events.
- Edges E exist from vertex u to vertex v if the event associated with u immediately precedes the event associated with v in temporal order.
- The edge weight $wt(u, v)$ is the sum of number of times the event associated with u immediately precedes the event associated with v in the event log.

This graph G will henceforth be referred to as a *follow graph*. The adjacency matrix A for G is defined by:

$$A_{uv} = \begin{cases} wt(u,v), & \text{if an edge exists between } u \text{ and } v \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

It is recommended to prune the graph by deleting edges with low weights. This is analogous to supplying a minimum support threshold in frequent itemset mining. After pruning,

remove singleton vertices, else each singleton vertex will be classified as its own cluster.

Pruning is based on a user-defined minimum threshold. The value can be chosen specific to a dataset. As the weights of the edges represent the number of times a pair of events appear adjacent to each other in the event log, larger datasets can afford a higher pruning threshold. The considerations for the pruning threshold include a trade-off between computation time, robustness, and usefulness of results. A minimum threshold = 0 (no pruning) leads to the longest computation time and the most number of events being clustered. This ensures the lower frequency events (e.g., critical events) will appear in the results, but the clusters may contain a lot of noise and extraneous events that do not truly represent a frequent pattern. On the other hand, a higher threshold will find frequent patterns more efficiently, but the results may miss key events that happen more sparingly. We recommend trying a range of thresholds and then settle down on a manageable outcome.

B. Community Detection

Since the follow graph is innately able to capture the associations between events, our next goal is to extract the events which co-occur frequently from the graph. For that purpose, we use community detection methods [16]. The idea of community detection is to partition a graph such that the vertices within a cluster are "close" to one another with respect to some measure of distance or similarity. When using edge weights in our follow graph as a similarity metric, a community detection method provides the set of events that co-occur frequently enough to be considered as "belonging to the same community."

Community detection assigns vertices of a graph into clusters using a similarity measure. We use community detection to get an initial clustering of events.

One popular category of community detection algorithms are called hierarchical clustering, which iteratively creates clusters [16]. We chose to utilize these types of algorithms because the hierarchical structure provides both a sense of similarity and dissimilarity between events. In addition, it uses a connectivity-based similarity metric which can be measured by attributes of the graph.

For our experiments, we used the hierarchical clustering algorithm by Clauset et al. [17].

Our chosen similarity metric, modularity, quantifies the strength of a graph division into partitions. It is a popular metric for determining the stopping criteria of hierarchical clustering algorithms [16]. Defined in (2), modularity quantifies the significance of a partition compared to a completely random partition. Originally defined for unweighted graph [18], A represents the adjacency matrix consisting of 1s and 0s. $m = \frac{1}{2}\sum_{uv} A_{uv}$ is the number of edges in the graph. $k_u = \sum_v A_{uv}$ is the degree of vertex u. The δ-function $\delta(i, j)$ equals 1 if $i = j$ and 0 otherwise. The modularity ranges from 0 to 1, with 0 indicating randomness and values

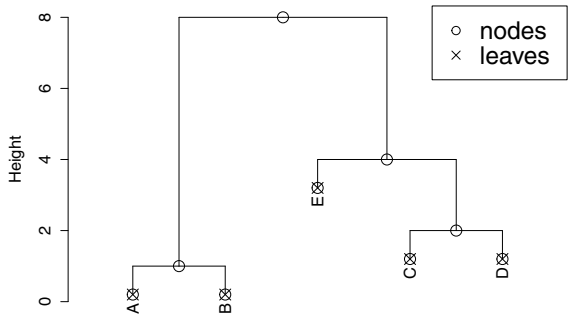

Fig. 2. Simple dendrogram example.

greater than 0 indicating significance of the partition [18].

$$Q = \frac{1}{2m}\sum_{uv}\left[A_u v - \frac{k_u k_v}{2m}\right]\delta(c_u, c_v) \tag{2}$$

Modularity can be generalized for weighted graphs [19] by defining A as an adjacency matrix of edge weights (1), then updating the values in (2) accordingly.

By the construction of the follow graph, the edge weights indicate strong associations. Therefore, the edge weights should be used in the calculation of the modularity for the community detection algorithm. Using the edge-weighted modularity allows for a better representation of the partition in the follow graph.

The result of hierarchical clustering can often be represented with a dendrogram.

C. Subclustering

Although community detection returns clusters of events, subclustering can provide a higher level of granularity, especially within large clusters. Subclustering finds the highly associated subclusters by using the dendrogram to determine distances. Events that are highly associated will be within a close distance i.e, a small height difference, specified by a threshold parameter. Subclustering finds the frequent event patterns with high co-occurrence in the event log.

Dendrograms are tree diagrams customarily used to illustrate and represent hierarchical clustering [16]. Dendrograms are a specific type of binary tree structure where the parent nodes represent a cluster containing only its children. Fig. 2 shows a simple dendrogram structure. Leaves are special nodes that contain no children. Clustering can be done by making a cut on the dendrogram at a specific height.

A dendrogram can be created such that each vertex is a leaf. Each merge in the hierarchical clustering algorithm represents a node in the dendrogram. The height of each node and leaf is equivalent to the iteration number in the hierarchical clustering algorithm where the merge of two events occurred.

The overall dendrogram provides multiple levels of granularity to find relationships among events and subclusters. Since any cut in the dendrogram can be considered a cluster, there are numerous possibility for clustering.

To perform subclustering, a distance metric needs to be defined for events (repesented as leaves) in the dendrogram.

Require: $dendrogram, heightfraction$
1: Let $threshold$ = $heightfraction\times$ height($dendrogram$)
2: **for** each node k **do**
3: Let partition P_k = leaf descendants of k
4: $distance_k$ = height(k) − $\min_{p\in P_k}$(height(p))
5: **if** $distance_k \leq threshold$ **then**
6: partition P_k is a candidate subcluster
7: **end if**
8: **end for**
9: **if** $P_i \nsubseteq P_j, \forall j \neq i$, where P_i, P_j are candidate subclusters **then**
10: P_i is a subcluster
11: **end if**
12: **return** $subclusters$

Fig. 3. Subclustering algorithm

The subclustering algorithm uses a notion from phylogenetics called cophenetic value [20] as a distance metric. The cophenetic value for two leaves in a dendrogram is equivalent to the lowest height of their common ancestors.

The subclustering algorithm is presented in Fig. 3. A user-defined parameter determines the maximum allowable distance between any two events to be considered within a subcluster. If all the descendant leaves of a node are within the threshold height difference, then it is a candidate subcluster. Since the nodes are self-containing, if a parent node is a subcluster candidate, all of its descendant nodes will also be candidate subclusters. Therefore, subset subclusters should be removed (lines 9-10 in Fig. 3).

IV. EXPERIMENTAL RESULTS

We have three sets of event log data from various sources. The graph-based clustering methodology is applied to each of the datasets and compared with association mining for frequent itemsets. The association mining algorithm used in these experiments is the `Apriori` algorithm [6].

A. Datasets

The three datasets will henceforth be labeled as Dataset A, Dataset B, and Dataset C, respectively. Dataset A is provided by an oil and gas company. The data contains one month of event logs with 1458 unique events and 38,899 observations. Dataset B is provided by a hydro power generation station. The data contains thirteen months of event logs with 6009 unique events and 149,977 observations. Dataset C is provided by an offshore wind farm. The data contains four years of event logs with 172 unique events and 39,039 observations. Due to confidentiality agreements, any identifying information is removed from all three datasets.

B. Comparison of Approaches

The `Apriori` association mining algorithm cannot be directly applied to event log data. Association mining traditionally requires an input of baskets of items, whereas an event log is essentially one long sequence of items. Therefore, we need to basketize the data by cutting the event log into a series of time intervals. Each time interval will constitute one basket.

TABLE I
DATASET A: FREQUENT ITEMSETS RESULTS

time interval	10 min	20 min	30 min
baskets	4464	2232	1488
support	0.0035	0.007	0.0105
unique events	275	261	258
frequent itemsets	265,677	401,842	1,324,921
closed frequent itemsets	2864	5687	9947
coverage	0.897	0.891	0.889

Results for the three datasets' association mining frequent itemsets are presented in Tables I, II, and III. In association mining, support is a user-defined minimum threshold. It is the minimal fractional value of all baskets that the itemset must appear in to be considered a "frequent" itemset. The events must meet the specified support to qualify as a candidate item in frequent itemsets. Therefore, not all events in the event log will be included in the results of association mining algorithm. The number of unique events contained in the frequent itemsets found by the `Apriori` algorithm are described in the experimental results. Coverage is represented by the fraction of observations in the data log which are contained in the set of unique events as defined. It can be thought of as an information measure. Although there are many frequent itemsets, closed frequent itemsets are often used for analysis instead. Closed frequent itemsets contain no supersets with the same support [8].

The subclustering results are presented in Tables IV, V, and VI for the three datasets. Coverage and the number of unique events for sublclustering results are defined in a similar manner as for association mining. The exception is that we consider subclusters rather than frequent itemsets.

There are two input parameters for the graph-based clustering approach: pruning threshold and height fraction. Pruning threshold is a user-defined mininmum threshold for edge weight in the graph. Height fraction is a fractional multiplier which gives the maximum distance two leaves in a dendrogram can be considered as "close," and therefore, a subcluster. A maximum height fraction value of 1 would consider all leaves in the dendrogram to be "close."

The two parameters in association mining, time interval and support, are analogous to the two parameters in subclustering, height fraction and pruning threshold. Time interval and height fraction both determine which events can be considered "close" in occurrence, whereas support and pruning threshold are both minimum thresholds for determining what can be considered "frequent" itemset in the association mining case, or "frequently co-occurring" subcluster.

TABLE II

DATASET B: FREQUENT ITEMSETS RESULTS

time interval	10 min	20 min	30 min
baskets	58,320	29,160	19,440
support	0.001	0.002	0.003
unique events	608	565	542
frequent itemsets	1,082,393	1,113,293	2,216,476
closed frequent itemsets	2823	3428	4474
coverage	0.729	0.716	0.709

TABLE III

DATASET C: FREQUENT ITEMSETS RESULTS

time interval	1 day	3 days	7 days
baskets	50,955	16,972	7,267
support	0.0006	0.0018	0.0042
unique events	68	46	35
frequent itemsets	208	202	241
closed frequent itemsets	207	201	235
coverage	0.978	0.946	0.894

V. DISCUSSION OF RESULTS

A. Choosing Subclustering Parameters

Examining the subclustering results in Tables IV, V, and VI, it can be observed that higher pruning thresholds can be tolerated for larger datasets. For example, Dataset B contains nearly 150,000 observations. Compared to association mining (Table II), a pruning threshold (Table V) of 4 or 5 can provide similar coverage levels. Lower pruning thresholds can result in better coverage. Conversely, for Datasest A and C, a pruning threshold of 0 or 1 is required to obtain similar coverage levels as association mining. We recommend choosing the highest possible pruning threshold that allows an acceptable coverage level.

TABLE IV

DATASET A: GRAPH-BASED CLUSTERING RESULTS

Pruning threshold		Height fraction 0.05	0.1	0.15	0.2	0.25	0.3
0	unique events	720	806	855	906	949	1001
	subclusters	284	262	238	218	207	203
	coverage	0.748	0.867	0.914	0.939	0.958	0.969
1	unique events	381	407	429	453	469	482
	subclusters	145	142	132	129	125	110
	coverage	0.638	0.743	0.809	0.862	0.879	0.893
2	unique events	277	291	308	321	339	350
	subclusters	106	103	98	92	78	70
	coverage	0.596	0.666	0.734	0.783	0.825	0.846
3	unique events	222	231	243	259	272	285
	subclusters	86	82	78	76	69	66
	coverage	0.586	0.613	0.678	0.746	0.789	0.822
4	unique events	212	220	230	241	351	262
	subclusters	82	78	77	75	68	62
	coverage	0.596	0.627	0.690	0.737	0.767	0.797
5	unique events	184	191	200	211	224	228
	subclusters	71	69	67	63	59	54
	coverage	0.572	0.582	0.643	0.683	0.737	0.747

TABLE V

DATASET B: GRAPH-BASED CLUSTERING RESULTS

Pruning threshold		Height fraction 0.05	0.1	0.15	0.2	0.25	0.3
0	unique events	3165	3462	3658	3812	3958	4106
	subclusters	1168	1072	958	875	835	788
	coverage	0.812	0.878	0.914	0.933	0.946	0.958
1	unique events	2268	2397	2501	2585	2684	2748
	subclusters	842	797	753	711	616	513
	coverage	0.776	0.841	0.870	0.888	0.903	0.915
2	unique events	1780	1876	1979	2027	2076	2120
	subclusters	667	636	603	573	495	410
	coverage	0.753	0.802	0.846	0.861	0.873	0.882
3	unique events	1461	1546	1598	1652	1689	1722
	subclusters	552	529	502	439	387	332
	coverage	0.723	0.776	0.807	0.825	0.838	0.847
4	unique events	1265	1337	1393	1430	1465	1484
	subclusters	479	458	440	396	344	306
	coverage	0.702	0.752	0.783	0.802	0.817	0.824
5	unique events	1152	1213	1268	1304	1327	1343
	subclusters	438	414	402	360	321	290
	coverage	0.686	0.734	0.762	0.784	0.796	0.802

TABLE VI

DATASET C: GRAPH-BASED CLUSTERING RESULTS

Pruning threshold		Height fraction 0.05	0.1	0.15	0.2	0.25	0.3
0	unique events	52	62	68	76	80	90
	subclusters	21	21	21	20	20	20
	coverage	0.892	0.914	0.921	0.930	0.932	0.938
1	unique events	50	58	63	72	75	80
	subclusters	20	20	20	20	20	19
	coverage	0.890	0.909	0.917	0.927	0.928	0.930
2	unique events	38	46	50	54	62	63
	subclusters	14	14	14	14	14	14
	coverage	0.871	0.894	0.898	0.902	0.908	0.908
3	unique events	36	39	44	48	53	56
	subclusters	13	13	13	12	12	12
	coverage	0.872	0.883	0.892	0.896	0.900	0.902
4	unique events	36	37	44	47	51	54
	subclusters	13	11	11	11	11	11
	coverage	0.869	0.873	0.889	0.893	0.896	0.898
5	unique events	33	35	41	45	50	54
	subclusters	14	10	10	10	10	10
	coverage	0.862	0.865	0.882	0.888	0.891	0.898

The height fraction is proportional to the maximum distance between two events to be considered as part of the same subcluster. Higher height fractions will provide higher coverage, but the average number of events per subcluster (the number of unique events divided by the number of subclusters) will increase. Therefore, the subclusters will be less frequently co-occurring compared to lower height fractions. We recommend choosing a lower height fraction that provides an acceptable coverage level and gives suitable subcluster sizes (which depends on the dataset).

B. Coverage versus Frequent Pattern Sets

We will consider the frequent pattern sets for association mining and subclustering to be closed frequent itemsets and subclusters, respectively. When comparing the number of

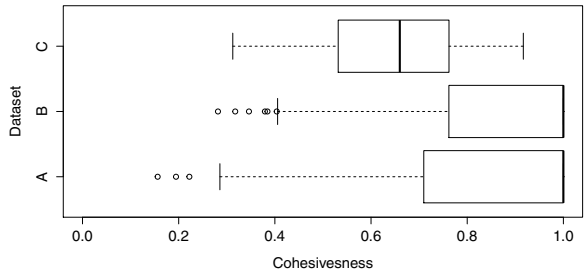

Fig. 4. Cohesiveness of subclusters

closed frequent itemsets versus the number of subclusters for similar coverage values, there are substantially less number of subclusters. Therefore, similar amounts of information are conveyed using less frequent pattern sets in subclustering. This provides more manageable results which can be more easily interpreted and applied to practical applications.

C. Quality of Subclusters

In addition to comparing subclusters with frequent itemsets, we established a metric to quantify the performance of subclusters found in the graph-based clustering method. We are able to use the graph structure to establish *cohesiveness*.

Cohesiveness of subcluster c, defined in (3), measures how tightly clustered c is by examining the weights of all edges connected to vertex u within c. The value can range from 0 to 1. A high cohesiveness indicates that events within a subcluster contain many edge connections with each other compared to edge connections with other events. Since the edges of our graph were determined by co-occurrence, high cohesiveness should correspond with frequent co-occurrence.

$$cohesiveness(c) = \frac{\sum\limits_{u,v \in c} wt(u,v)}{\sum\limits_{u,v \in c} wt(u,v) + \sum\limits_{\substack{u \in c \\ w \notin c}} wt(u,w)} \quad (3)$$

Fig. 4 shows boxplots for the cohesiveness of the subclusters in each dataset. The chosen parameters (pruning threshold, height fraction) for each dataset are as follows: Dataset A (1, 0.25), Dataset B (4, 0.1), and Dataset C (0, 0.25). For Datasets A and B, at least half of the subclusters obtained using the specified parameters in our method have a maximum cohesiveness value of 1. For Dataset C, half of the subclusters have a cohesiveness greater than 0.66. Therefore, the subclusters obtained from our method correctly grouped events that were frequently co-occurring.

VI. CONCLUSIONS

Results from the proposed graph-based clustering solution have been compared with traditional association mining of frequent itemsets. The subclusters are able to accurately capture the co-occurrence relationships. This is established by comparing the coverage levels with the association mining results. This information is relayed through substantially less frequent sets. It also does not require basketization, which could lose some information from the original event log data.

Graph-based clustering produces manageable and easily interpretable results because each subcluster indicates a frequent co-occurring set of events. Since events cannot be repeated in a different subcluster, each subcluster can represent one distinct collection to be further examined.

REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Record*, vol. 22, no. 2, pp. 207–216, June 1993.

[2] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[3] R. Alves, D. S. Rodriguez-Baena, and J. S. Aguilar-Ruiz, "Gene association analysis: a survey of frequent pattern mining from gene expression data," *Briefings in Bioinformatics*, vol. 11, no. 2, pp. 210–224, 2010.

[4] P. N. Nohuddin, F. Coenen, R. Christley, C. Setzkorn, Y. Patel, and S. Williams, "Finding interesting trends in social networks using frequent pattern mining and self organizing maps," *Knowledge-Based Systems*, vol. 29, pp. 104 – 113, 2012.

[5] N. Jiang and L. Gruenwald, "Research issues in data stream association rule mining," *SIGMOD Record*, vol. 35, no. 1, pp. 14–19, Mar. 2006.

[6] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 487–499.

[7] B. Liu, W. Hsu, and Y. Ma, "Mining association rules with multiple minimum supports," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '99. New York, NY, USA: ACM, 1999, pp. 337–341.

[8] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Database Theory ICDT99*, ser. Lecture Notes in Computer Science, C. Beeri and P. Buneman, Eds. Springer, 1999, vol. 1540, pp. 398–416.

[9] Y. Koh and N. Rountree, "Finding sporadic rules using apriori-inverse," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, T. Ho, D. Cheung, and H. Liu, Eds. Springer, 2005, vol. 3518, pp. 97–106.

[10] C.-K. Chui, B. Kao, and E. Hung, "Mining frequent itemsets from uncertain data," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Z.-H. Zhou, H. Li, and Q. Yang, Eds. Springer, 2007, vol. 4426, pp. 47–58.

[11] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, vol. 1, Oct 2007, pp. 305–312.

[12] C. C. Aggarwal and J. Han, Eds., *Frequent Pattern Mining*. Springer, 2014.

[13] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu, "Mining access patterns efficiently from web logs," in *Knowledge Discovery and Data Mining. Current Issues and New Applications*, ser. Lecture Notes in Computer Science, T. Terano, H. Liu, and A. Chen, Eds. Springer, 2000, vol. 1805, pp. 396–407.

[14] C. Ezeife and Y. Lu, "Mining web log sequential patterns with position coded pre-order linked wap-tree," *Data Mining and Knowledge Discovery*, vol. 10, no. 1, pp. 5–38, 2005.

[15] R. Vaarandi, "A breadth-first algorithm for mining frequent patterns from event logs," in *Intelligence in Communication Systems*, ser. Lecture Notes in Computer Science, F. Aagesen, C. Anutariya, and V. Wuwongse, Eds. Springer, 2004, vol. 3283, pp. 293–308.

[16] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 35, pp. 75 – 174, 2010.

[17] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, p. 066111, Dec 2004.

[18] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, p. 026113, Feb 2004.

[19] M. E. J. Newman, "Analysis of weighted networks," *Physical Review E*, vol. 70, p. 056131, Nov 2004.

[20] F. J. R. Robert R. Sokal, "The comparison of dendrograms by objective methods," *Taxon*, vol. 11, no. 2, pp. 33–40, 1962.